Relational Algebra

Relational Algebra

- Last time: started on Relational Algebra
  - What your SQL queries are translated to for evaluation
- A formal query language based on *operators*

Unary operator

Binary operator

Selection operator

- Input: a relation
- Output: a relation containing a subset of the tuples from the input relation
  - That satisfy a certain condition

bid	name	color
101	Misty	red
102	Pearl	blue
103	Speedy	blue

selection

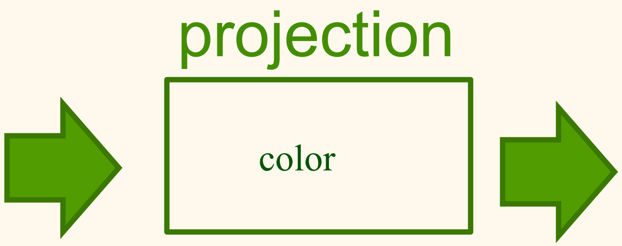
bid = 101

bid	name	color
101	Misty	red

Projection operator

- Input: a relation
- Output: a relation containing a subset of the columns from the input relation

bid	name	color
101	Misty	red
102	Pearl	blue
103	Speedy	blue



color
red
blue

- Relations are sets, so duplicates removed!!

Cross Product

bid	name	color
101	Misty	red
102	Pearl	blue

sid	bid	day
22	101	9/10/13
58	102	9/11/13

cross product

???

In mathematical notation

```
SELECT   S.sname
FROM     Sailors S, Reserves R
WHERE    S.sid=R.sid AND R.bid='101';
```

$$\pi_{S.sname}(\sigma_{R.bid=101 \wedge R.sid=S.sid}(R \times S))$$

What does this buy us?

- Explicit workflow ("where the tuples go")
- Can start thinking about implementation:
  - Need an implementation for each of the boxes
  - Maybe can reorder and/or combine some of them for better results?

Now let's make it more formal

- So far: intuitive "boxes and arrows" presentation
- Now: more formal mathematical specification
- Useful because:
  - If you implement these operators, have to know precisely what you're implementing
  - If we want to reorder some of them, need to know (**prove**) that it's safe

Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance
- A query is a sequence of operators
- Operators need to compose (can feed output of one as input to the next one)
  - So both input and output relations need to be sets
  - **Set** relational algebra
    - Next time: bag relational algebra (duplicates allowed)

Preliminaries

- Schemas of operator outputs determined by schemas of inputs
  - E.g. selection -> same schema
  - Projection -> subset of columns

- Positional vs. named-field notation:
  - Positional notation (arguably) easier for formal definitions, named-field notation more readable.
  - We mostly use named-field notation in this course

Relational Algebra

- The "core" operators:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Join* ($\bowtie$) Technically redundant, but very handy
  - *Set operators:* $\cup$, $\cap$, $-$

Example Instances

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

- "Sailors" and "Reserves" relations for our examples.

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Selection

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

- Selects rows that satisfy *selection condition*.
- *Schema* of result identical to schema of input relation.

$$\sigma_{rating>8}(S2)$$

Selection – formal definition

- Formally, let c be a selection condition
  - i.e. a Boolean combination (using AND and OR) of terms of the form "*att op const*" or "*att1 op att2*", where
    - *att, att1, att2* are attribute names
    - *op* is =, !=, <, >, <=, >=
    - *const* is a value from the domain of the attribute in question
- Then

$$\sigma_c(R) = \{t \in R \mid c \text{ is true for } t\}$$

Projection

- Deletes attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the input relation.
- Eliminates duplicates (relations are sets)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname,rating}(S2)$$

age
35.0
55.5

$$\pi_{age}(S2)$$

Projection – formal definition (1)

- For a tuple t , let $t[A]$ denote the restriction of tuple t to exactly the attributes in A

- Suppose $A = \{a_1, a_2, \cdots a_k\}$
- Suppose $t(a)$ denotes value of tuple t for attribute a
- Then $t[A] = t(a_1) \cdot t(a_2) \cdots t(a_k)$
- The dot represents concatenation

Projection – formal definition (2)

- Let R be a relation and A a subset of the attributes of R

- Then $\pi_A(R)$ is defined as

$$\{t[A] \mid t \in R\}$$

Cross-Product

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
  - *Conflict*: Both S1 and R1 have a field called *sid*.

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

- *Renaming operator*: $\rho(C(S1.sid \to sid1, R1.sid \to sid2), S1 \times R1)$

Cross product –formal definition

- Not that hard!

$$R \times S = \{(r \cdot s) \mid r \in R \wedge s \in S\}$$

Join operator

- Very common case: cross product followed by selection that "connects" attributes from both relations
- Handy to define a shorthand operator for it
  - But technically don't need to

Joins

- ***Condition Join***: $R \bowtie_a S = \sigma_a(R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently

Joins

- *Equi-Join*: A special case of condition join where the join condition contains only ***equalities***.

$$S1 \bowtie_{S1.sid=R1.sid} R1$$

A handy shortcut

- If the join condition is equality, unwieldy to carry two copies of column in result
- Add a projection to remove one duplicate column
- Shortcut (per your textbook): assume the projection without specifying it
  - i.e. retain only one copy of each column where join condition has equality

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{S1.sid=R1.sid} R1$$

Natural joins

- $\bowtie$ symbol without condition
- Equality join on all fields with common name in both tables
- And retain only one copy of each such field
- If want to join on only *some* of the common attributes, need to specify condition explicitly

Union, Intersection, Set-Difference

- All of these operators take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - `Corresponding' fields have the same type.

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

Formal definitions

- Easier and easier!

$$R \cup S = \{t \mid t \in R \vee t \in S\}$$

$$R \cap S = \{t \mid t \in R \wedge t \in S\}$$

$$R - S = \{t \mid t \in R \wedge t \notin S\}$$

Confusion alert!

- Union vs. Join
- These are NOT the same thing
- Let's make sure we understand the difference!

A few examples

- A few more examples to get the hang of it

Find names of sailors who've reserved boat #103

- Solution 1: $\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$

- Solution 2: $\rho(Temp1, \sigma_{bid=103} Reserves)$

$$\rho(Temp2, Temp1 \bowtie Sailors)$$

$$\pi_{sname}(Temp2)$$

Find names of sailors who've reserved a red boat

- Information about boat color only available in Boats; so need an extra join:

$$\pi_{sname}(\sigma_{color=red}(Boats \bowtie Reserves \bowtie Sailors))$$

- A more efficient solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid}(\sigma_{color=red}B)) \bowtie R) \bowtie S)$$

A query optimizer can find this, given the first solution!

Find sailors who've reserved a red or a green boat

- Can identify all red or green boats, then find sailors who've reserved one of these boats:

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}\ Boats))$$

$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

- Can also define Tempboats using union! (How?)

Find sailors who've reserved a red and a green boat

- Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):

$$\rho\ (Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie \text{Reserves}))$$

$$\rho\ (Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie \text{Reserves}))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Additional operators

- The basic operator toolkit is the one you've seen
- But can be handy to introduce extra operators that are technically redundant
  - Joins are an example
  - Intersection is actually redundant too (try to express it using the others)